CS2323: Computer Architecture

Lab-8 Report

Ganji Varshitha
AI20BTECH11009

Input and Output Formats

Input is a text file which consists of test cases, one line for one instruction machine code.
For example:

```
007201b3
00720863
00c0006f
00533623
100004b7
00c50493
```

Output is another text file which has the assembly code for each corresponding instruction.
Output for above instruction looks like:

```
add x3, x4, x7
beq x4, x7, L1
jal x0, L1
sd x5, 12(x6)
lui x9, 0x10000
L1: addi x9, x10, 12
```

Parsing Input file

After opening input.txt file in r mode using open method, and simultaneously opened output.txt file in w mode, we read each line in input file and parse the line to get the desired instruction assembly code and write to output.txt file.

Storing the output instruction in an object

I defined a class called instr_str which has 2 arguments: data and offset.
After parsing the line, we need to get the hex number to decimal and get the opcode for the instruction and call the desired format function using eval(opcode[op]) where opcode contains mapping of opcode to instruction format which is also function it should be directed to using eval().

Getting number from few bits inside the machine code

This is done by implementing a function named bits which takes the number(decimal format) and start position along with number of bits to be considered from there.
Code goes like this:

((1 << num_bits) - 1) & (num >> (start-1))

num >> (start-1) shifts the number such that start position bit is LSB.
(1 << num_bits) - 1) gives 0s followed by num_bits number of 1s.
Logical and operation extracts the desired bits.

Basic Approach

After calling a particular function based on format of the instruction like R,I,S,B, etc. , we have to compute rd, rs1,rs2 numbers from the specified position.
For R format, we have funct7 field and mapped funct7 with 2 sets of instructions which are further mapped by funct3 field values.
Similarly, for I - format, we have to take care of the immediate field values as it is signed integer.
For branch and jal type of instructions, offset should be left shifted by 1 bit.
Also the bits are shuffled completely and signed bit should be taken care of by negating the value of that bit.

Offset and Labels

Each instr_str object for each machine code has offset values but consists of non zero values for branch and jal instructions.
I used a dictionary for line number of such instructions to its offset values.
Line no. of branch = line number of B/J instruction - $\lfloor \frac{\text{offset}}{4} \rfloor$
After writing all the strings to output.txt file, we read all the lines in the output again and modifies the lines of branch instruction and line of branch using a list of branch names and removing the branch label from the list after using for one line,offset pair.

Edge cases

We need to take care of unconditional loops where offsets is 0. Which is solved using if else conditional statements.
Immediate values of srai and srli should be in the range of 0-31 and consider last 5 bits.(as they have same funct3 value)